NETWORK DATA HANDLING SYSTEM

Thomas Marill

Computer Corporation of America

Prepared for:

Army Research Office-Durham
Advanced Research Projects Agency

31 January 1973

COMPUTER CORPORATION OF AMERICA

DATACOMPUTER PROJECT
SEMI-ANNUAL TECHNICAL REPORT

August 1, 1972 to January 31, 1973

Contract No. DAHC04-71-C-0011
ARPA Order 1731

Submitted to:

Advanced Research Projects Agency
1400 Wilson Boulevard
Arlington, Virginia 22209

Attention: Program Management

37

Computer Corporation of America

575 Technology Square
Cambridge
Massachusetts 02139

617-491-3670

DATACOMPUTER PROJECT
SEMI-ANNUAL TECHNICAL REPORT

August 1, 1972 to January 31, 1973

Contract No. DAHC04-71-C-0011
ARPA Order 1731

Submitted to:

Advanced Research Projects Agency
1400 Wilson Boulevard
Arlington, Virginia 22209

Attention: Program Management

Unclassified
Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY *(Corporate author)*
Computer Corporation of America

2a. REPORT SECURITY CLASSIFICATION
Unclassified

2b. GROUP
NA

3. REPORT TITLE
Network Data Handling System

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*
semiannual TR: 1 August 1972 - 31 January 1973

5. AUTHOR(S) *(First name, middle initial, last name)*
Thomas Marill

6. REPORT DATE
1973

7a. TOTAL NO. OF PAGES
31p

7b. NO. OF REFS

8a. CONTRACT OR GRANT NO.
DAHC04 71 C 0011.

b. PROJECT NO.

c.

d.

9a. ORIGINATOR'S REPORT NUMBER(S)
NA

9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*
9816.2-A

10. DISTRIBUTION STATEMENT
Approved for public release; distribution unlimited.

11. SUPPLEMENTARY NOTES
None

12. SPONSORING MILITARY ACTIVITY
U. S. Army Research Office-Durham
Box CM, Duke Station
Durham, North Carolina 27706

13. ABSTRACT
This report describes the activities for the period 1 Aug 1972 - 31 Jan 1973.

During this reporting period, the activity on the project has centered on development of the first software release, initial system demonstration, coordination with potential users, and work on a global weather data base.

14. KEY WORDS

Datacomputer Project
Computers
Data Systems
Large-scale Data Systems
Multiple computers

DD FORM 1 NOV 68 1473 REPLACES DD FORM 1473, 1 JAN 64, WHICH IS OBSOLETE FOR ARMY USE.

Unclassified

Computer Corporation of America
575 Technology Square
Cambridge, Massachusetts 02139

DATACOMPUTER PROJECT
SEMI-ANNUAL TECHNICAL REPORT

August 1, 1972 to January 31, 1973

This research was supported by the Advanced Research Projects Agency of the Department of Defense and was monitored by the U.S. Army Research Office-Durham under Contract No. DAHC04-71-C-0011. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Advanced Research Projects Agency or the U.S. Government.

Table of Contents

1. Overview

The goal of the project continues to be the development of a shared, large-scale data system to serve the needs of the ARPA community.

The system under development may be viewed as a box that performs the functions of data storage and data management on behalf of multiple computers simultaneously connected to the box. The computers can access the box directly over local links or remotely over the Arpanet; in either event, a standard notation--datalanguage--is used to access the box. Inside the box there is a medium-scale computer, secondary storage for data-staging, and an on-line tertiary storage device.

Some of the ideas underlying the development are as follows: (a) The approach has the effect of pooling many users' data-storage requirements into a single system. As a consequence, economies of scale can be realized by employing ultra-large tertiary storage devices (Unicon, TBM, others) that have a high price tag but very low per-bit cost. (b) The effective use of remote storage systems requires that the data-management functions themselves be performed within the system. (It is uneconomical to ship entire data files over a network when, as is usually the case, only small portions of the files are needed at a given time.) (c) In a system such as this, problems of data security become more tractable. The reason is that the data storage and data management functions are segregated into a separate box that responds interpretively to a limited set of commands. In the more conventional environment, arbitrary user processes run in the same computer as the data functions, making it hard if not impossible to guarantee data security. (d) The system language--datalanguage--is being designed for use in the Arpanet as a standard means of access to remotely located data. It contains features specifically designed for

sharing data among programs that operate on different machines, for describing a broad class of data structures, and for allowing arbitrary subsets of large files to be selected efficiently at run-time.

For the prototype system being developed at CCA in Cambridge, a dedicated PDP-10 TENEX with a billion bits of secondary disk storage is being used, and plans for the addition of a large-scale tertiary store are being formulated. The software system running at CCA will also be run at NASA/Ames, where a tertiary store (the Unicon 690) has recently been installed. Service will be offered out of both the east and west coast facilities, which will be in communication with one another for purposes of mutual backup.

During this reporting period, the activity on the project has centered on development of the first software release, initial system demonstration, coordination with potential users, and work on a global weather data base.

In the software area, a first version of the system was implemented. This version--release 0/8--contains much of the structure that will make up the final system (some two dozen modules), though many modules exist as yet only in primitive form. Release 0/8 handles a very small set of the full data-language; in the future increasingly larger sets will be made available through subsequent releases. A limited experimental service for cooperative users is being initiated now using 0/8, and the system is being used for demonstration and system tests.

As an initial demonstration of release 0/8, a month's worth of the world's weather was loaded into the system. A natural-language front-end program was implemented which accepts arbitrary English questions about the weather and attempts to answer these. This front-end translates the English into datalanguage requests, which then access the datacomputer system to generate the output. The demonstration was run for the three days of the International Conference on Computer Communications in October, during which time the database was kept up-to-date on a daily basis.

Participation in the Weather Database Working Group has continued, as well as coordination with the dozen or so groups that have expressed interest in becoming users of the system.

2. Design Activities

2.1 Datalanguage

The first two releases of the language were specified. The first was designed to provide basic facilities for dealing with a file of weather observations. This file has a hierarchical structure which is described explicitly in datalanguage. Requests for retrieval specify the desired observations by content, select items of interest from those observations, and arrange them into a form acceptable to the user program. The second release extends the language to provide basic updating capabilities--adding to, and deleting files. While our intentions for development of the language make these releases look rather primitive, the exercise of specifying them completely has been revealing.

Rules for a simple class of description-to-description mappings were written for the first release. These define the meaning of A=B when A and B are not identical. Defining A=B through such a descriptive mapping has the advantage that it specifies the result of the assignment, even for complex structures, rather than specifying a procedure by which the assignment is carried out. This allows the datacomputer the freedom to optimize the operation.

It was discovered that automatic mapping mechanisms are complex to specify. Thus, if the system maps one elaborate structure into another, either this must be performed according to very involved rules, or it must handle only rather similar descriptions. The former would be difficult to use; the latter may be limited in application (though still not necessarily devoid of practical interest).

However, it is thought that any particular mapping is not difficult to specify, and may frequently deviate in simple ways from straightforward standard mappings. Thus attention will next turn to the design of a facility for users to specify and dynamically modify description mapping functions.

In specifying these releases, progress was made in determining the meaning of names in datalanguage requests. The meaning of a name (i.e., whether X means a particular X, a set of Xs, as well as which X or set of Xs) depends in part on the context in which it appears. Again, this problem was solved for certain special cases for the first release. The rules for recognition in contexts have been formalized for these simple cases, and the exercise of designing them has forced us to accept that (a) a more precise model for sets and objects is required to complete the language design, and (b) recognition of names and manipulation of contexts at the directory level (currently avoided except in trivial cases) cannot be specified in the language independent of directory access characteristics (searches are expensive).

Feedback from users of the first release has affected our attitudes about datalanguage syntax and process synchronization. The experience with these users has revealed some of the distinctions between the servicing of people through a language and the servicing of remote, asynchronous programs. We have come to feel that the syntax of datalanguage is too elaborate--that it is complicated by mechanisms natural for people and troublesome for programs trying to generate requests. This has caused us to step back and investigate other kinds of syntax appropriate to our semantics.

The other result of experience with users--synchronization difficulties--has brought about some changes already and seems to be initiating some more. Datacomputer responses, in the form of messages on the datalanguage output port, are being supplemented by codes designed for program interpretation. In addition, messages are being added to aid in synchronization. Also, a very simple user-to-datacomputer synchronization message has been added. These improvements, however, seem inadequate, and it is likely that something more general--perhaps a datacomputer protocol--will be developed. This is also expected to feed back into the language design.

Finally, we have done some investigation of the implications of data sharing for the language, and begun to design more powerful description facilities than were originally envisioned. First, we have begun to develop the concept of the virtual container, which is a data object that is not stored anywhere, but could be produced by executing some datalanguage request. Virtual containers behave like real containers, and a user need not be aware that they are virtual. Each virtual container description corresponds to some collection of data as some particular user cares to view it.

There are several implications of this concept. One is that user programs can be independent of the stored organization of data. (Earlier attempts at datalanguage design provided some independence of the representation of items, but relatively little independence of structure). Thus data can be reorganized as needs change, without changing existing programs (except as dictated by altered performance). Another implication is that programs need be aware only of the portions of

data structures which concern them. Finally, virtual containers can be thought of as a way of recording data relationships. Consider a program which commonly retrieves information from related containers in files A and B. If it enters into the datacomputer a description of a virtual container which 'contains' the information of interest from A and B, then information about the relationship of A and B has been captured. This is now in descriptive form, under the management of the system, rather than isolated in a request or built into a user program. This information could be useful to those administering the databases, as well as to system programs.

Another key concept developed in this period is description partitioning. We know that the data description contains many levels of information, and that each level is useful for different operations. The lowest level contains specifications for the storage or transmission medium in which the container exists. The highest level contains only that information which is required for understanding the data. At sundry levels in between are logical access paths, choices of representation, information related to validity checking, protection and security, operational considerations (e.g. age, expiration date, backup/recovery requirements), and so on. In transferring data from one storage medium to another, but otherwise leaving it unchanged, only the lowest level of description is affected. In adding indices, pointers, or inverted file keys, only the middle levels are affected. Some users only concern themselves with the top level. These parts of the description must exist in the system as separate objects, and they must be accessible independently.

2.2 Software System

In regard to software system design, very little activity ha- taken place beyond what was reported in the previous semi-annual report. Essentially, what is being implemented at this time is the design given in Working Paper No. 5 (Feb. 29, 1972).

3. Software Implementation

Software implementation, started in the last reporting period, is progressing. The culmination of this period's activity was the generation of release 0/8, which ran the ICCC demonstration in October (see Section 4, below). This release is quite primitive. It handles only fixed-length ASCII strings, provides for only one user at a time (multiple users must have multiple copies), does not support tertiary storage, has poor debugging facilities, etc. Even so, this release is attracting a number of potential users. A draft of a user-oriented guide to 0/8 has been prepared.

3.1 Request Handler

A Request Handler for the first release of the language was programmed and debugged. It is developed along the lines of the system architecture outlined in Working Paper No. 5. Most of the code, and particularly the module-level design of the program, is expected to remain a part of the system, providing a foundation for further work.

This Request Handler incorporates an inverted file retrieval system, as discussed in Working Paper No. 2. Sufficient file maintenance routines exist to support the current level of language development. Parameters in the user's data description specify whether a field is to be an inverted file key. Construction of the inverted file, and use of it in expressions referencing such a key, are invisible to the user. Therefore, datalanguage requests referencing the file are independent of the file creator's choice of keys.

The inverted file system, however, is not complete. Remaining to be implemented are extensions to handle variable-size containers, changing or deleting of key values, range retrieval, expressions involving hierarchical relations, large files, and known algorithms to optimize massive file updates.

Parsing of the language implemented thus far is straightforward, with the exception of recognition rules for names. These are implemented in a module (CX) separated from the parser proper (LP). Requests which can be executed immediately are passed by LP to a command executor (CO); those which must be analyzed and compiled are transformed into a parse tree, which is input to the Source Analyzer (SA).

The CX module is concerned with disambiguation of partial names and relating names to data object instances or sets. Of the two name spaces in the datacomputer, CX operates on only one. This is the set of names defined in open data descriptions. CX should also function on the set of names defined in the currently relevant part of the directory. Extension of CX to operate on the directory name space will be attacked in the coming year.

SA is intended to choose among alternate paths of access to data referenced by a request. In order to do so, it must first expand the parse tree statement of the request into procedure expressed at the proper level. As it does so, it produces an intermediate language graph, which is designed to allow easy re-structuring of the request as decisions are made.

Currently, SA makes no significant strategy decisions. It does decide whether or not to use inverted file techniques, but according to an extremely crude heuristic. However, it expands the tree, makes loops explicit, and moves certain operations outside of the loops. The graph it outputs consists of operations on elementary containers (such as assignment and comparison), loop operators, and open/close operators. These are intended to map straightforwardly into tuples (see below).

The graph is then input to a code generator (CG) which outputs code for a software-implemented 'machine' (TI) whose instructions operate on data containers. The instructions are called tuples.

Implementation of CG has proved complicated, and approaches alternate to the original one are under consideration. The difficulties are related to low-level optimization problems, which are expected to become significant when the current restrictions on data structures are removed in the coming year. The basic problem is to know which pointers into a data structure are going to be available unconditionally when control arrives at a certain point. (Having solved this, one can go further, asking whether previously executed code should be constrained to obtain certain pointers, whether it would naturally do so or not.) When a required pointer is not available, tuples are generated to obtain it.

One solution under consideration involves visualizing the process of obtaining pointers into a data structure as distinct from the process of operating on the elements of the data

structure. The former depends, in most cases, only on the data structure itself. The latter is a function of the current request and is responsible for most of the complexities in program flow. The idea is to compile separate tuple strings for each such process, controlling them as cooperating sequential processes or as linked co-routines in a single process.

The 'machine' which executes these tuples was defined and implemented in this period. Programming and debugging was convenient, since tuple executors could be worked on independent of one another.

In thinking about the implementation of the interpreter, we began to develop a model for the container structures the interpreter is operating on. We would like to complete this model and use it in designing the lower levels of the data description language.

3.2 Services

The services portion of the software consists of a set of subroutines which are available to the Request Handler. The architecture of services has been described in Working Paper No. 5. With the exception of those calls dealing with tertiary storage, a complete (though primitive) first set of service routines has been implemented for release 0/8. Re-design of some modules and enhancements of others will be undertaken during 1973. The activity breaks down into four parts:

Storage Manager

The following have been programmed: core buffer management, logical-physical address mapping, storage allocation, device-dependent I/O functions.

I/O Manager

External I/O functions have been programmed to handle the case of ASCII strings.

Directory System

Functions for building and accessing a tree structured file directory as well as for storage of corresponding data descriptions and storage maps have been implemented.

Supervisory Functions

Routines for handling errors, software interrupts and user and database initialization have been developed.

4. Initial System Demonstration

In response to a general request by L.G. Roberts at the 1971 IPT principal investigators' meeting that natural-language database oriented demonstrations be given at the ICCC conference, such a demonstration was produced, and represented the first external demonstration of the datacomputer system.

The demonstration was held on October 24-26 at the Statler Hilton, Washington, D.C., as part of the Special Projects activity of the International Conference on Computer Communications. The handout used for the demonstration is given in Appendix A.

The datacomputer software that was used for the demonstration was release 0/8, which is discussed above. This release was the first version of the system to be used outside of CCA. A natural-language program translates English questions into release 0/8 datalanguage, which is then passed on to the datacomputer software for data retrieval. The natural-language front-end and release 0/8 communicate with each other through the IMP, thereby providing a realistic test of 0/8 and also allowing the natural-language program to be transported to some other installation.

4.1 Data Acquisition

A month's worth of global weather information from the ETAC weather file was loaded into the datacomputer system (using disk storage). The database was kept up-to-date on a daily basis during the three days of the ICCC conference.

It had been originally planned for GWC to send the raw data to ETAC by network transmission for reformatting, and for ETAC to send the reformatted data to CCA by network transmission. Due to a variety of technical problems, some of the data was finally received at CCA by network transmission and some by air freight; some was received from ETAC and some directly from GWC. The latter was converted directly at CCA.

Specifically, 7 tapes were received from ETAC by air freight and 1 by network; 8 tapes were received from GWC by air freight and 10 by network. During the three demonstration days, update data came from GWC over the net, so that the previous day's weather was available each morning.

4.2 Datacomputer Software

The datacomputer software used in the demonstration was release 0/8 discussed above (Section 3).

4.3 Natural-Language Front-End

A natural language front-end which understood English questions about the weather was developed. This system is based on work done by Professor T. Winograd of the MIT Artificial Intelligence Lab.

The Winograd system was converted from MIT-AI LISP to BBN-LISP to run at CCA's TENEX. The parsing routines needed no modification. A new vocabulary dealing with the weather was added, with much of the old functional vocabulary (words such as "be", "what") retained. The major modification was to have the system generate datalanguage instead of the PLANNER requests generated by the original Winograd system.

In the Winograd system, an internal semantic description of the meaning is constructed as the sentence is understood. That description is then translated into PLANNER code. This is an easy translation since PLANNER is a goal-oriented system, embodying a sophisticated matcher. The translation of a request into datalanguage is substantially more difficult, especially considering the elementary level of datalanguage available at the time of the ICCC demonstration. This problem was solved by using a two stage translation process. The set of relations derived in understanding the sentence are converted to LISP expressions, which are evaluated. These expressions put their pieces of information onto the property lists of variables which are then gathered together to form the datalanguage request.

For example, the sentence "Has it rained in Boston lately?" has the following internal semantic form:

```
RSS5: (MARKERS= ( SYSTEMS  RELATION)
                PARSENODE=
                (NODE1)
                PLAUSIBILITY= 0 RELATIONS=
                (RSS4 RSS2 ( MORE  PRECIP RSS1 0)
                      ( WEATHER  PRECIP RSS1 TSS1))
                RSSNODE= RSS5 SYSTEMS= ( SYSTEMS)
                VARIABLE= EVX1)
```

RSS4 and RSS2 describe the time and place relations. This is converted to the following LISP program:

```
(PROG (EVX1)
      ( CITY TSS1 ((BOSTON MASSACHUSETTS)))
      ( TIME TSS1 ((293 300)))
      (SETQ EVX1 ( WEATHER  PRECIP EVX1 TSS1))
      (SETQ EVX1 ( MORE  PRECIP EVX1 0))
      (PUT (QUOTE EVX1)
           (QUOTE BIND)
           EVX1)
      (RETURN EVX1))
```

When executed this program produces the following datalanguage request:

```
FOR STATION WITH  (REGION EQ 'MASSACHUSETTS' AND CITY EQ 'BOSTON')
     FOR ANSWER.ANS, OBSERVATION WITH DATE GE '293' AND DATE LE '300'
          ANS.LP1 = '(' ANS.CITY = STATION.CITY ANS.RP1 = ')'
          ANS.DATE = OBSERVATION.DATE
          ANS.LP2 = '(' ANS.DATA =  PRECIP ANS.DATA1 = '    '
          ANS.DATA2 = '    '  ANS.RP2 = ')' END END
```

The parse diagram of this sentence is:

```
(((HAS IT RAINED IN BOSTON LATELY)
  (CLAUSE MAJOR TOPLEVEL QUEST POLAR POLR2 ACTV ITRNS)
  (RSS20)
  ((HAS (HAVE VB AUX TRANS V3PS PRESENT QAUX))
   ((IT)
    (NG SUBJ PRONG DEF NS)
    (OSS20)
    ((IT (PRON NS SUBJ OBJ))))
   ((RAINED)
    (VG V3PS)
    NIL
    ((HAS (HAVE VB AUX TRANS V3PS PRESENT QAUX))
     (RAINED (VB ITRNS PAST EN MVB))))
   ((IN BOSTON)
    (PREPG)
    (RSS19)
    ((IN (PREP PLACE))
     ((BOSTON)
      (NG OBJ DEF PROPNG CITY NS)
      (OSS22)
      ((BOSTON (PROPN NS CITY))))))
   ((LATELY)
    (ADJG ADV TIMW)
    (RSS17)
    ((LATELY (ADV VBAD TIMW))))))
 ((OSS22 X12 ((BOSTON MASSACHUSETTS)))))
```

A sample word definition is:

```
RAIN:
  (FEATURES (NOUN NS MASS VB INF ITRNS)
            SEMANTICS
            ((NOUN (OBJECT (MARKERS: (#PRECIP)
                                     PROCEDURE:
                                     ((#WEATHER #PRECIP *** *TIME)
                                      (#MORE #PRECIP *** 0)))))
             (VB ((ITRNS (RELATION (RESTRICTIONS: ((SMSUB (#WEATHER
                                                           #GENERAL)))
                                                  PROCEDURE:
                                                  ((#WEATHER #PRECIP
                                                             *** *TIME)
                                                   (#MORE #PRECIP *** 0)
                                                   ))))))))
```

4.4 Sample Datacomputer File

The following datalanguage port descriptions are used to retrieve data from the datacomputer.

```
CREATE CHECK PORT LIST
        PLACE STRUCT
        LP1 STR (1)
        CITY  STR (22)
        RP1 STR (1)
        BSN STR (6)
        LP2 STR (1)
        REGION STR (22)
        RP2 STR (1)
        END  PLACE STRUCT
END  CREATE CHECK

CREATE ANSWER PORT LIST
        ANS STRUCT
        LP1 STR (1)
        CITY  STR (22)
        RP1 STR (1)
        DATE STR (3)
        LP2 STR (1)
        DATA STR (4)
        DATA1 STR (4)
        DATA2 STR (4)
        RP2 STR (1)
        END  ANS STRUCT
END  CREATE ANSWER
```

The following is the datalanguage description of the file as it is stored in the datacomputer.

```
CREATE WEATHER FILE LIST
   STATION STRUCT
      BSN          STR(6),              I=D
      CITY         STR(22),             I=D
      REGION       STR(22),             I=D
      WORLD        STR(22)
      OBS          LIST (31)
       OBSERVATION        STRUCT
        DATE      STR(3)
        TEMPERATURE        STRUCT
           MIN    STR(4)
           MAX    STR(4)
        END  TEMPERATURE STRUCT
        PRECIP    STR(4)
        WINDS     STRUCT
           SPEED           STR(4)
           GUSTS           STR(4)
           DIRECTION       STR(4)
        END  WINDS STRUCT
        VISIBILITY         STR(4)
        CLOUDS    STR(4)
        GENERAL STR(4)
        PRESSURE           STR(4)
        END  OBSERVATION STRUCT
      END  STATION STRUCT
   END  WEATHER FILE
```

The following data is a sample record--the first one in the file. In the datacomputer it is stored without the carriage returns needed to make it print neatly.

```
010100ANDOYA                                NORWAY                    EUROPE
275
276 278 279    0   2   0  12   9   758883023
277 279 283    4  23  52  11   8   788882982
278
279 274 281    0  23  42  12   9   788882976
280 268 270    4  14   0   9  19   627772983
281 276 283    0  28  69   7   9   688882986
282 269 273    4  10   0   9  22   727772975
283 278 284    0  13  37   9   7   766882972
284            0  28   0   5   1   80006   0
285 276 280    0   9   0   3  12   666682936
286 273 275    0  15   0   3   7   877782960
287 278 282    0  13  28  11  14   626662981
288 269 271   28  30   0   6   7   888882908
289 275 281    0  21  45   7   9   688882976
290 268 269    0  20   0   8  22   622782960
291 283 283    0  11   0   7  47   100013002
292 274 274    0  11   0   1  47   500012996
293
294 272 272    0   5   0   7  47   400012949
295
296
297 273 273    0   0   0   1   6   80007   0
298 271 275    0   0   0   1   2   80007   0
299 275 275    0   6   0   7  31   30000   0
300
301
302
303
304
305
```

5. Investigation of Tertiary Storage Devices

An investigation of the characteristics of commercially available, high-density storage devices was undertaken during this period, and a somewhat more detailed evaluation of the UNICON was made.

5.1 System Characteristics

There are two basic approaches being taken today in commercially available, high density, tertiary storage devices: laser recording and magnetic recording. The method of laser recording developed by Precision Instruments, for example, involves vaporizing 3-4 micron holes in rhodium-backed mylar strips to write data. To read data, the same laser at lower intensity is used, and variations in reflected laser light are detected by a split photodiode which also serves to aid in tracking.

Other means of using the laser are being investigated but are not well developed yet. For example, Carson Labs is developing a method of storing data using volumetric holograms retained at the color centers of cryogenically cooled sodium-doped potassium chloride crystals.

In magnetic recording technology, Ampex Corporation has applied video recording techniques to obtain a packing density of 10^6 bits per square inch. Using standard 2 inch wide video tape, but recording data in duplicate, Ampex gets 4.5×10^{10} bits per reel. Other systems using high-density magnetic recording techniques are being developed by Grumman Data Systems, Control Data, and others.

Each type of system currently available has various parameters that must be taken into account. A number of these are given in Table 1 for four specific devices: Precision Instruments Unicon, Ampex TBM, International Video Corporation HID recorder, and Grumman Mass Tape.

Table 1

Characteristics of Four Tertiary Storage Devices

(as of 4/6/72)

	Unicon	TBM	IVC-HID	MASSTAPE
Capacity	$.7 \times 10^{12}$b	3×10^{12}b	9×10^{10}b	10^{12}b
Error Rate	1b in 1.5×10^{8}b	1b in 2.7×10^{11}	1b in 10^{9}	1b in 10^{9}
Access Times	See text	See text	See text	See text
Availability	95%	90% unit 99.9% system	99.9% unit quote, system unknown	unknown
Transfer Rate	3Mb/drum 6Mb total	6Mb/channel 36Mb max	8Mb	1.5Mb/channel 48Mb total
State of Development	Not ready for test	ready, tested	no system, only device	being tested & developed
Cost	$1.6M	$500K ($10^{11}$b) to $4.4M ($3.2 \times 10^{12}$b)	$50K each unit system unknown	$2M
Maintenance	unknown	simple replacement	for units, excessive for system unknown	unknown
Modularity	a dual device for degradable backup	excellent	for unit, none, for system, assumed	excellent
Storage Media	not updatable permanent	updatable read or write 2000 passes	updatable read or write 500 passes	updatable read or write 5000 passes

A critical dimension is that of access time. We have analyzed this in terms of seek time and skip-read time.

The seek time is the time required to position the device to prepare to read a given block of data. Since seek times will typically be greater than the time required to retrieve the data for a request, seek times become the important factor in determining system throughput.

Figure 1 shows seek time as a function of the number of bits separating the current position and the desired position. The times given assume a single controller. Multiple controllers can reduce the "system average" seek time by a factor of 2 to 32 depending on the device. It should be noted that this factor is not necessarily equal to the number of controllers because of interactions between the seek processes.

The skip-read-time (Fig. 2) refers to the time spent per block averaged over both skipped blocks and read blocks. The skip-read-time for each device is dependent upon the number of blocks skipped per block read (the skip ratio) and upon the block size chosen. For this analysis, the relevant block size is 2^{14} bits. Figure 2 shows the relationship between skip-read-time and skip-ratio for each of the 4 mass memory devices as well as the DEC RP02 disk.

The DEC RP02 disk has been included for comparison and as an example of the calculation procedure. The device characteristics are: time to read a page = 10 ms, rotational period = 25 ms, pages per cylinder = 50, cylinders = 200, access time to a new cylinder = between 12 ms and 60 ms (assumed linear). Thus reading all pages takes 10 ms per page. Reading between 1 and 50 pages per cylinder, ignoring latency optimization, takes (10 ms + 12.5 ms + (P/50) 12 ms) per page read. Between

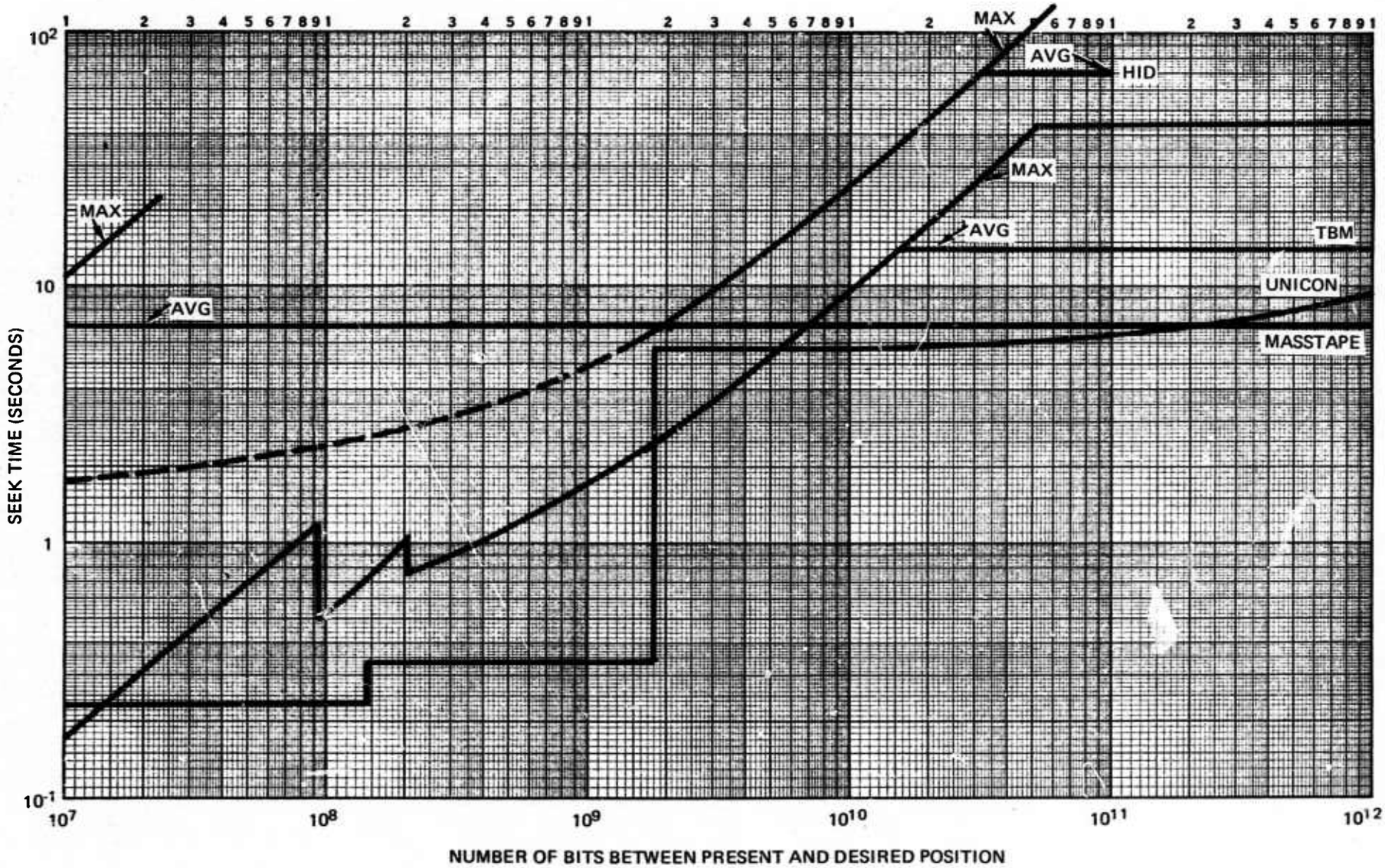

Fig. 1 - Seek Times for Tertiary Storage Devices

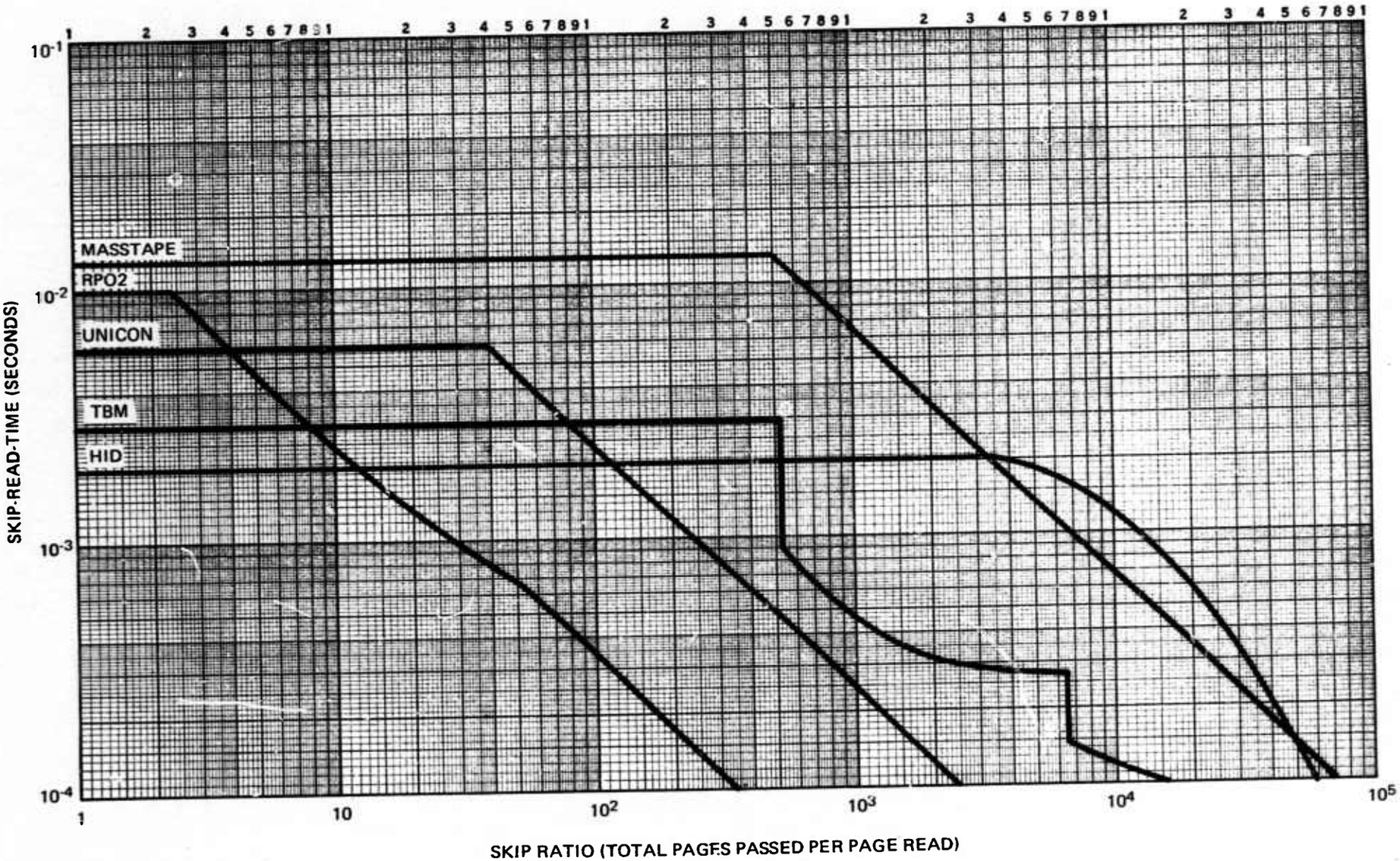

Fig. 2 - Skip-Read Times for Tertiary Storage Devices

1 and 200 pages per pack, the time per page read is 10 ms + 12.5 ms + (200/n) (60 ms-12 ms)/200 + 12 ms. The time per page read is then divided by the total number of pages passed to determine the skip-read-time.

For the Unicon, 7½ pages per track have been assumed. A record 2-1000 tracks (15 - 7.5×10^3 pages) away can be accessed in 240 ms. One 1000-11440 tracks (7.5×10^3 - 8.6×10^4 pages) away can be accessed in 340 ms.

In the TBM, reading speed of 5 ips (5.6 Mb/sec) is used for searches less than 8 inches (512 pages) away. Capstan drives of 83 and 248 ips are used for distances less than 96" and 200" respectively. An overhead of 434 ms to read a page after search is currently imposed by the software. This can be reduced to 234 ms.

Complete access time data on the HID recorder was not obtainable; however, 5 seconds is taken to slow from search speed, read a block, and return to search speed. About 10^9 bits (6.1×10^4 pages) are traversed in this operation. Assuming a 5" separation before it is practical to search at a higher speed than the reading speed, the curve would be approximately as shown.

For the Masstape, the relevant data are: 16 tracks per tape (each recorded individually), 2.2×10^7 bits (1.34×10^3 pages) per track, with a read and search rate of 1.2 Mb (73.4 pages) per second. Time to switch tapes is .6 sec.

5.2 Unicon Evaluation

A somewhat more detailed evaluation of the Unicon was undertaken (with the assistance of Professor Haim Haskal of Tufts). The findings were as follows.

5.2.1 Main Causes of Error

A. Material Problem.

The recording medium used by PI is a thin coating of rhodium deposited by sputtering on a transparent mylar base. The quality of the mylar base is very important--no bumps or pits--because it affects the recording material quality. After deposition the completed strip is checked for reflection and transmission by a laser system at a "macroscopic scale". The test of material quality at a "microscopic scale" is done in the laser recorder itself by the so-called read-during-write verification. This verification only checks for strong reflection from the rhodium coating corresponding to a logical "0". No read-during-write system exists which verifies the "1" bits as burnt. Until such a system is developed (PI is presently trying to develop it), one does not really have assurance that the data have been properly recorded. Thus, one should take one more drum revolution, thus increasing access time, to read after write. By PI's claims, the number of "bad spots" per strip is about 250 which would represent a maximum raw error of 0.6 in 10^7 bits. This represents a very small defect density and would make the material acceptable provided that the "bad spots" are properly identified and by-passed.

B. Dust Problem.

This is a general problem one faces in all high density recording systems. In the PI recorder in spite of the 0.5 micron dust filters a considerable amount of dust enters the machine and contaminates the drums and the strips. In fact a so-called "tent effect" has been observed in which the strip is lifted off the drum locally,

enough to deform the strip to make recording or reading impossible. It would require particles of the order of 1/2 to 1 mil to produce this effect. Much remains to be improved on the air system; for that reason NASA/Ames plans to locate the laser memory in a clean room.

C. Tracking Problems.

In the PI system every time a strip is placed on a drum its skew is computed and stored in a register. This then serves as error information for the galvanometer. In addition, the galvanometer receives current tracking information from a split photodetector to correct for minute tracking errors. It is not clear why this is necessary and in fact R. Brown at NASA/Ames claims that considerable improvement in tracking was obtained by disconnecting the skew register.

During acquisition it is relatively straightforward to jump to an adjacent track and acquire on the first try. However as one ventures to further out tracks one requires several passes in order to acquire the right track. This is due to the error in the optical shaft encoder of the carriage.

Possible improvement can be realized by using a laser interferometer which can measure travel distances with an accuracy of 1-5 micro inches. (1 micrometer = 40 micro inches.) Such an instrument is currently being manufactured by Hewlett-Packard and was considered by R. Brown also. However, it was rejected because it can only cope in the velocities of travel of up to 12 inches/sec. In discussions with Hewlett-Packard they mentioned the possibility of doubling the velocity to up to 24 inches/sec. which would make it attractive for incorporation in the PI system. The laser interferometer would replace the optical shaft encoder and make acquisition possible for any track in the strip on the first try. Thus the memory would be capable of operating in a more random fashion.

5.2.2 Reliability and Maintenance

A. Argon Laser.

The argon laser is operated very conservatively, at about half its rated current, thus the plasma tube should last at least 3000-4000 hours. The major difficulty arises from the fact that with every tube change a major realignment is necessary.

B. Alignment.

By PI's own saying the alignment of the optical train requires about 8 hours of an experienced serviceman. This is due to two factors: the optical path is unduly long and thus more prone to misalignment by small angular variations or translations and also due to the lack of alignment aids to facilitate the procedure. There is no doubt the optical design could be much improved.

C. Depth-of-Focus.

In view of the tight tolerance on the depth of focus, $\pm$0.5 mil, there may be some question about long term changes in the drum bearings which may shift the focus. It may be desirable to use an air-bearing on the drum or at least to monitor the depth of focus by a capacitive probe or some other device.

6. Miscellaneous Activities

6.1 Network Usage and Analysis

As already noted (Section 4.1) a certain amount of experience in transferring data over the net was obtained in connection with data acquisition for the ICCC demonstration. This experience seemed to indicate that TIPS could not yet be used reliably for tape transmission and that tape transfers into our TENEX installation were very slow.

To shed further light on the problem a series of measurements was made* and distributed through the NIC. The results seem to indicate that the maximum transmission rate through the TENEX/network interface was on the order of 300K baud, and that at that rate, 100% of CPU time would be used.

6.2 Meetings and Conferences

The activities of technical coordination with prospective users and other interested parties has continued. Interactions have taken place with NSA, VELA, MITRE, MIT, NASA/Langley, LRL, and NASA/Ames. Technical presentations were given at SHARE in Toronto and as part of a panel "Data Sharing in Computer Networks" in San Francisco.

6.3 Weather Database Working Group

The third meeting of the WDBWG was at NASA/Ames on January 18, 1973. Plans for ETAC's initial use of the datacomputer were discussed. CCA will participate with ARPA and ETAC in preparing a paper on the CCA-ETAC Weather Data Base for publication.

---

* H. Murray, TENEX Bandwidth, Computer Corporation of America, Cambridge, Mass., Nov. 29, 1972. Available from Network Information Center as NIC 10428.

Appendix A

The following is a handout used to describe the CCA demonstration given at the International Conference on Computer Communications, Washington, D.C., October 24-26, 1972.

Computer Corporation of America

Demonstration of Datacomputer with Natural-Language Front-End

Overview

In this demonstration, a person at a computer terminal may type an arbitrary question, expressed in ordinary English, dealing with the weather. The system will attempt to understand the question and to find an answer. If successful, it will display the answer; if not, it will display an explanatory message.

The demonstration makes use of the resources of the datacomputer system and of a special natural-language front-end program operating at the dataconiputer.

Datacomputer

The datacomputer is a large-scale on-line data storage and data management system being developed by Computer Corporation of America for the Advanced Research Projects Agency. When in full operation, the datacomputer will have a storage capacity of over 10^{12} bits.

Remote computers in the Arpanet can send to the datacomputer information for storage, update information to be applied to existing files, and information-retrieval requests to be answered. Interaction with the datacomputer is in a uniform system of notation called *datalanguage*.

Natural-Language Front-End

For demonstration purposes, a natural-language program has been implemented. This program—a modification of a natural-language system developed by T. Winograd of MIT—accepts English-language questions dealing with the weather and translates these questions into datalanguage for input to the datacomputer.

The natural-language program accepts a sentenco from a terminal. It parses the sentence from left to right, building up an internal semantic description which embodies the meaning of the sentence. This description is then translated into datalanguage.

Two sentences with the same meaning but different form generate the same datalanguage request. For example, the two sentences:

- Did it rain in Boston two days ago?
- Was there any precipitation the day before yesterday in Boston?

both cause the following datalanguage request to be sent to the datacomputer:

```
FOR STATION WITH CITY EQ 'BOSTON'
   FOR ANSWER.ANS, OBSERVATION WITH DATE EQ '151'
      ANS.CITY = STATION.CITY
      ANS.DATE = OBSERVATION.DATE
      ANS.DATA = PRECIP
   END
END
```

Weather Database

Weather data is shipped over the Arpanet to the datacomputer in Cambridge, Massachusetts from the Environmental Technical Applications Center (ETAC) in Washington, D.C. ETAC maintains a file of all weather observations from around the world going back a decade, based on information shipped over the Arpanet from Air Force Global Weather Central (GWC) in Omaha, Nebraska. The file will be stored in its entirety on the datacomputer (along with other unrelated databases) and will be accessible on-line to computers on the Arpanet.

For purposes of demonstration, one month's worth of weather information is being used. The file contains daily observations from several thousand cities around the world.

Sample Questions

What was it like in Washington yesterday?
Was Cairo cloudy on Friday?
What was the barometric pressure in Paris last Tuesday?
How many times has it rained in Jerusalem this month?
How much?
What was the coldest day in Tokyo last week?
Which city in Uganda had the highest rainfall in October?
Was San Francisco colder than Los Angeles on the day before yesterday?
Where in Arizona did it rain yesterday?
Has any precipitation fallen in Massachusetts lately?
What was the temperature in New Delhi when the rainfall in Calcutta was more than 0.5 inches?
When was the temperature in Boston greater than 40 degrees?
Were there five cities in Texas whose high temperatures yesterday were more than 84 degrees?
What was the coldest place in Alaska recently?

Using the System

The following guide is a sample console session with the system. The symbol [LF] stands for line-feed, [CR] for carriage-return, and [ESC] for escape or alt-mode. User input is indicated here in lower case and underlined, system response is capitalized, and comments are italicized.

A character preceded by an up-arrow (↑) is a control character, which is typed by holding the control key (CTRL) and striking the character. Typing ↑A will cause the last character typed to be deleted. ↑X deletes the entire line, and ↑R retypes the line.

```
@r [LF]                                  Reset the TIP, terminal-dependent setup
                                         here.
@l[SP]31 [LF]                            Connect to CCA-TENEX.
LOGGER
R T OPEN
TENEX/CCA 2.29 EXEC 2.39                 You are now at the TENEX command level.
                                         TENEX prompts for input with an @-sign.
                                         At any time you can force a return to
                                         the command level by typing ↑C.
@login[SP]iccc [CR]
(PASSWORD) iccc [CR]                     Password will not print.
(ACCOUNT) iccc [CR]
JOB 5 ON TTY17 23-OCT-72 18:30
YOU HAVE A MESSAGE                       This message will contain some info
                                         pertaining to the CCA Weather Demo.
                                         To print the message, type:
@type[SP]message[ESC].TXT;1   [CR]
                                         Hitting the ESC key causes TENEX to
                                         complete the file name.  Hitting [CR] will
                                         cause the message to print on your console.
@run[SP]weather               [CR]       Run weather system.
READY                                    Indicates system ready to accept sentence.
what was it like in washington 3 days ago?
                                         Questions are terminated by "?"
THERE WAS LIGHT RAIN.
READY
what was the higheat temperture →TEMPERATURE in boston last week?
                                         System corrects spelling of temperature.
77 DEGREES, LAST TUESDAY.
READY
has it rained in arizona lately?
                                         "Lately" refers to the past seven days.
YES, 7 TIMES.
READY
where?
IN PHOENIX, TUCSON, AND YUMA.
READY
goodbye.                                 Tell system to return to TENEX command level.
GOODBYE.
@logout [CR]                             Log off TENEX.
@c [LF]                                  Close network connections.
R T CLOSED
```


Computer Corporation of America
575 Technology Square
Cambridge, Massachusetts 02139
617-491-3670